4 Séquence numéro 4 : interface. 2h TD, 4h TP

4.1 TD

4.2 TP - Interface KeyListener

A l'heure actuelle, votre projet doit être capable d'afficher un dongeon graphique. Si ce n'est pas le cas, vous pouvez vous resynchroniser avec l'avancé du TP en utilisant la solution en ligne à l'adresse suivante : `https://github.com/antoineTauvelENSEA/Whatever2D/tree/SeanceTPNumero2`

Le but de cette séance est de parvenir à déplacer sans animation le héros au sein de la map. Pour cela, nous allons compléter notre projet à l'aide du diagramme UML suivant.

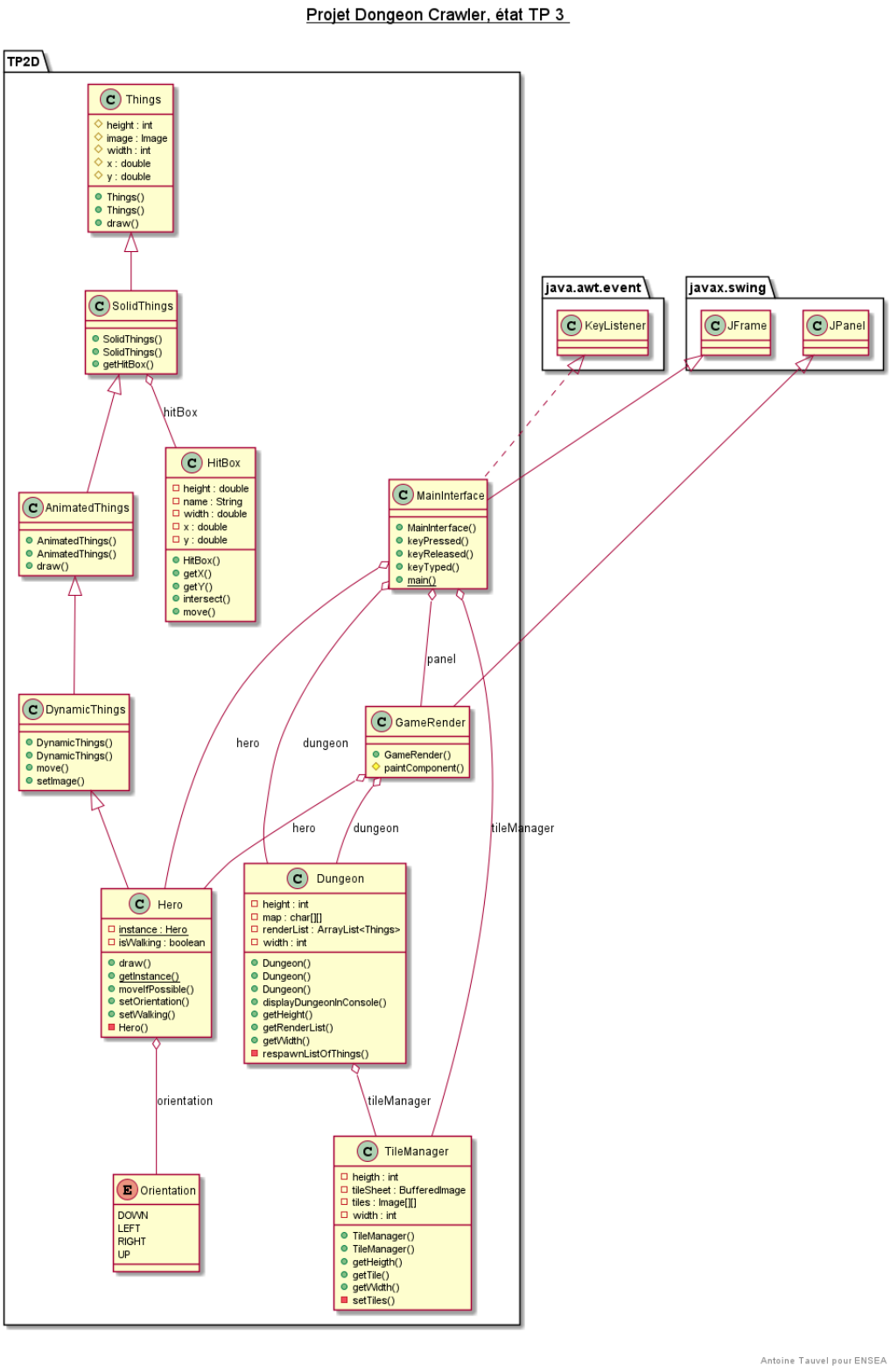

Figure 4.1: Le diagramme UML touche à sa fin.

4.2.1 Création de l'énuméré Orientation et modification de la classe Heros.

- Créez une classe énuméré Orientation comprenant 4 valeurs : DOWN, LEFT, UP, RIGHT, chacun associé à une valeur d'un entier nommé i (respectivement 0, 1, 2, et 3).
- Une telle classe doit avoir obligatoire un constructeur.
- Ajouter un getter `public int getI();`
- Ajouter un élément privé `Orientation orientation` au sein de la classe du héros.
- Ajouter au sein de la classe héros un setter pour la variable orientation.
- Ajouter au héros un attribut booléen `isWalking` faux par défaut, ainsi qu'un setter pour cet argument.

4.2.2 Interface KeyListener

- Au sein de la classe Maininterface qui gère notre affichage, rajouter l'interface KeyListener.
- Une erreur de compilation se produit. Pourquoi ?
- Surcharger les méthodes nécessaires à la bonne compilation.
- Au sein de la méthode `public void keyPressed(KeyEvent e)` testez l'affichage de `e.getKeyCode()`
- A l'aide d'une structure switch case comparant e.getKeyCode à `KeyEvent.VK_LEFT`, `KeyEvent.VK_RIGHT`, `KeyEvent.VK_DOWN` et `KeyEvent.VK_UP`, modifiez l'orientation du héros en fonction de la touche appuyé, ainsi que le booléen isWalking.
- En utilisant une autre méthode de l'interface, mettez isWalking à 0 lorsque la touche est relaché.

4.2.3 Affichage et déplacement du héros

Comme pour l'affichage du TP précédent, on doit surcharger la méthode `public void draw(Graphics g)`. Pour cela, on va utiliser la TileSheet suivante (ou une autre trouvé en ligne si vous le souhaitez).

Figure 4.2: La SpriteSheet du héros

Les images font 48x52 pixels, et sont séparé en hauteur de 100 pixels.

A l'aide de la documentation javadoc de la fonction `drawImage(image, dx1, dy1, dx2, dy2, sx1, sy1, sx2, sy2);` (dans laquelle d désigne la destination, s la source, le point 2 est en bas à droite, le point 1 en haut et à gauche), on va dessinez à la bonne place sur l'écran la bonne position du héros, sans animation pour le moment.

Ecrivez la fonction draw de la classe Héros. N'oubliez pas d'intégrer l'affichage du héros au sein de la classe GameRender qui constitue notre "moteur graphique".

Il est nécessaire de raffraichir l'image du héros si on veut visualiser le résultat de nos efforts. Pour cela, il suffit d'appeler la fonction repaint() au sein de la classe MainInterface, après la modification de l'orientation du héros. On supprimera cette ligne une fois l'orientation validé.

4.2.4 Animation du héros

A l'aide de la documentation javadoc de la classe `Timer` du framework swing, déclenchez un timer qui recalcule les affichages (comme plus haut) toutes les 50 millisecondes (framerate théorique de 20 Hz).

On saura lors de l'affichage du héros le temps courant grâce à la méthode statique `System.currentTimeMillis()`

Une variable locale entière "index" doit calculer l'image afficher. Cette image est le fruit de la division du temps courant par le temps entre deux frame (typiquement une centaine de millisecondes, dépendant de la vitesse que l'on souhaite), modulo le nombre d'image disponible. Ces deux éléments devraient dans une bonne programmation être des variables finale du héros.

Le héros doit avoir l'air d'avancer seulement si `isWalking` vaut 1.

- Ajoutez un timer. Vérifiez son fonctionnement à l'aide d'un println que vous supprimerez ensuite
- Calculez la variable index au sein de la méthode draw du héros.
- Modifiez la méthode draw pour obtenir l'affichage de l'animation.

4.2.5 Déplacement du héros

Il est nécessaire dans cette partie d'agrandir le petit dongeon du début...

On y est presque. Si `isWalking` vaut 1, on doit essayer de bouger le héros. Pour cela, on doit connaitre son environnement pour tester la présence de mur ou d'ennemi.

Pour vous récompenser, voici une aide pour l'écriture de la fonction moveIfPossible (On a du coder une nouvelle fonction `move` sur notre classe HitBox) :

(Bien sur, vous pouvez coder vous même cette fonction...)

```
public void moveIfPossible(double dx, double dy, Dungeon dungeon){
        boolean movePossible=true;
        this.getHitBox().move(dx,dy);
        for (Things things : dungeon.getRenderList()){
            if (things instanceof SolidThings){
                if(((SolidThings) things).getHitBox().intersect(this.getHitBox())){
                    movePossible=false;
                    break;
                }
            }
        }

        if (movePossible){
            this.x=x+dx;
            this.y=y+dy;
        }
        else{
            this.getHitBox().move(-dx,-dy);
        }
    }
```

Listing 9: Solution au déplacement du héros dans un environnement contraint

- Codez la méthode move au sein de la classe HitBox.
- Codez la méthode moveIfPossible au sein du Héros
- Au sein de l'interface KeyListener, rajouter un appel à la méthode moveIfPossible d'un certain déplcement (typiquement 5 à 10 pixels).

Félicitations ! Il ne nous reste plus qu'à rajouter une gestion du niveau.